WebGl

And threejs

Mkdir “lesson one”

Echo > “index.html”

! -> press enter to add html boiler plate

Added <script src="./script.js"></script>

Into the <body> tag of html added a console.log() to the js script.

Apparently you can just drag and drop an html file to you browser and it will load what it looks like.

Did this to see our console.log()

Then we downloaded threejs master file from the .org site.

Then we pulled out the three.min.js script from the build folder into our folder with the other two scripts (the html and js)

Then we added the same line but BEFORE our script. So looks like:

 <script src="./three.min.js"></script>

    <script src="./script.js"></script>

From this we have access to the variable THREE. Most classes are what are in three.js

Console.log(THREE) to see the object and classes we have available.

Now to create first scene.

We need 4 elements to create a first scene.

Scene to contain Objects

Objects

Camera

Renderer

Scene is like a container, we put objects, models, lights

const scene = new THREE.Scene();

now we create objects. They can be anything like primitive geometries, imported models, particles, lights, etc.

to create a visible object, we call that a Mesh. It is a combination of a geometry and a material. We are going to make a red cube with boxgeometry and meshbasicmaterial.

const geometry = new THREE.BoxGeometry(1, 1, 1)

now create the mesh

const material = new THREE.MeshBasicMaterial({ color: "#ff0000" })

# hex goes red, green blue. So ff says add red and the 0000 means have no green or blue.

Or use

({ color: 0xff0000 })

This is a three.js color class.

Now actually add the mesh together

const mesh = new THREE.Mesh(geometry, material);

Now add the mesh to the scene

scene.add(mesh);

now create the camera. Render is from one camera.

Need to add two parameters to a camera

Field of view – vertical vision angle, in degress, called fov (field of view). Example 75

Aspect ratio – width of the render divided by the height of the render. We do not have a renderer yet but we can use temp values

//sizes

const sizes = {

  width: 800,

  height: 600,

};

// camera

const camera = new THREE.PerspectiveCamera(75, sizes.width / sizes.height);

scene.add(camera);

now we need to provide a renderer. A scene through your renderer point of view.

We are going to draw into a canvas using webGL.

A render is a scene through your camera point of view. Result drawn on a canvas. Canvas is an html element in which you can draw stuff. Three.js will use webGL to do this. We can create it or let three.js do it for us.

Now we create the <canvas> in the html.

<body>

    <canvas class="webgl"></canvas>

    <script src="./three.min.js"></script>

    <script src="./script.js"></script>

  </body>

Now we need to fetch this canvas from the html DOM by using

//get our canvas from the html DOM

const canvas = document.querySelector('.webgl')

when you resize the renderer, it will resize the canvas.

In the renderer.render you need to provide a scene and a camera.

const canvas = document.querySelector(".webgl");

const renderer = new THREE.WebGLRenderer({

  canvas: canvas,

});

renderer.setSize(sizes.width, sizes.height);

renderer.render(scene, camera);

but all we see on the webpage is a black square BECAUSE our camera is actually inside the cube in the middle. So, we need to move the camera back some

to move anything we can use the position, rotation, and scale properties.

To move object we have to move in x, y and z axis. We move camera.

In three js, the y is up & down, x is left and right, and z is toward and back.

So if we want to move the camera backwards, we have to change the z value.

So we add

const camera = new THREE.PerspectiveCamera(75, sizes.width / sizes.height);

camera.position.z = 3;

scene.add(camera);